Corso: **Tecnico di Reti e Sistemi informatici**
Anno formativo 1998/99

Modulo: **Analisi - Programmazione - Linguaggi**

Dispensa n° 2

ELEMENTI DI PROGRAMMAZIONE

A cura di **Sergio Fumich**

Fondazione Enaip Lombardia - CSF di Crema
Via Lago Gerundo 30/e, 26013 Crema (Cr) - ☎ 0373-200826
E-mail: crema@enaip.lombardia.it
E-mail dell'Area Tecnologie Informatiche: enaip@chizzoli.it

1. Algoritmi.

Con il termine algoritmo si intende, in termini grossolani, un testo che specifica una serie di operazioni, la cui esecuzione rende possibile la risoluzione di un determinato problema. Più genericamente si può anche dire che un algoritmo è un insieme di prescrizioni per effettuare un dato compito.
L'esperienza quotidiana suggerisce numerosi esempi di algoritmi: una ricetta di cucina, le istruzioni per l'uso di un elettrodomestico, le indicazioni per riempire un modulo, le regole per eseguire una operazione aritmetica.

1.1 Operazioni.

Un algoritmo prescrive l'esecuzione di operazioni. Una operazione è un "qualche cosa" che può essere eseguito, oppure una "azione" che può essere compiuta, oppure un "evento" che si può far accadere. "spostare una sedia", "sommare un numero ad un altro", "aggiungere un certo quantitativo di parmigiano al risotto" sono operazioni.
Le operazioni che un algoritmo può prescrivere possono essere di natura assai differente, ma devono avere caratteristiche ben precise.
Ogni operazione deve avere termine entro un intervallo di tempo finito dall'inizio della sua esecuzione.
Ogni operazione deve produrre, se eseguita, un effetto osservabile e che possa essere descritto assegnando lo "stato" immediatamente dopo tale esecuzione.
Ogni operazione deve produrre lo stesso effetto ogni volta che venga eseguita a partire dalle stesse "condizioni iniziali".

1.2 Sequenze di esecuzione.

L'esecuzione di un algoritmo da parte di un esecutore (uomo o macchina) si traduce in una successione di operazioni che vengono effettuate nel tempo. L'esecuzione di un algoritmo, cioè, evoca un processo sequenziale, una serie di eventi che occorrono uno dopo l'altro, ciascuno con un inizio ed una fine ben identificabili.
Per descrivere in qualche modo il processo sequenziale evocato da una determinata esecuzione di un algoritmo, elencheremo, una dopo l'altra, tutte le istruzioni eseguite, nell'ordine di esecuzione. Tale lista di istruzioni è detta una sequenza di esecuzione dell'algoritmo in esame.
Il processo evocato da un algoritmo può essere fisso o no, cioè può essere o non essere sempre lo stesso ad ogni diversa esecuzione.
Il seguente algoritmo:

```
Cerca la corretta aliquota IVA sulla tabella
Moltiplica l'importo netto per l'aliquota trovata
Somma il risultato all'importo netto
```

dice come calcolare l'importo di una fattura. Esso è composto da tre "istruzioni" che devono essere eseguite in successione. Le operazioni che si eseguono ed il loro ordine non cambiano qualunque sia la merce da fatturare, l'aliquota da applicare o l'entità dell'importo netto.
In tale caso la sequenza di esecuzione, essendo una sola, coincide in pratica con il testo dell'algoritmo stesso. Ma ciò non è sempre vero, i casi più frequenti sono quelli in cui uno stesso algoritmo può evocare più processi sequenziali differenti a seconda delle condizioni iniziali.
Il seguente algoritmo è una versione modificata del precedente per tener conto della possibilità di una merce non soggetta a IVA:

```
SE la merce da fatturare è soggetta a IVA
     ALLORA
          cerca la corretta aliquota IVA sulla tabella
          moltiplica l'importo per l'aliquota trovata
          somma il risultato all'importo netto
     ALTRIMENTI tieni conto solo dell'importo di partenza
```

Questo algoritmo può produrre due differenti sequenze di esecuzione.
Prima sequenza - caso della merce da fatturare soggetta ad IVA:

```
Cerca la corretta aliquota IVA sulla tabella
Moltiplica l'importo per l'aliquota trovata
Somma il risultato all'importo netto
```

Seconda sequenza - caso della merce esente da IVA:

```
Tieni conto solo dell'importo di partenza
```

Nelle situazioni descritte l'algoritmo produce un numero di sequenze di esecuzione finito e noto a priori. Altre situazioni più complesse possono generare un numero di sequenze d'esecuzione descritte da uno stesso algoritmo, non solo non noto a priori, ma addirittura esso può essere infinito.
Il seguente algoritmo per effettuare una telefonata:

```
Solleva il ricevitore
componi il numero
SE qualcuno risponde
      ALLORA conduci la conversazione
      ALTRIMENTI deponi il ricevitore e RIPETI L'INTERO PROCEDIMENTO
```

descrive la sequenza seguente di esecuzione se l'interlocutore risponde al telefono al primo tentativo:

```
Solleva il ricevitore
Componi il numero
(Qualcuno risponde)
Conduci la conversazione
```

Se la telefonata riesce al secondo tentativo si ha la sequenza:

```
Solleva il ricevitore
Componi il numero
(Nessuno risponde)
Deponi il ricevitore
Solleva il ricevitore
Componi il numero
(Qualcuno risponde)
Conduci la conversazione
```

In generale l'algoritmo descrive infinite sequenze di esecuzione, corrispondenti al fatto che la telefonata riesca al terzo, quarto, quinto, ... tentativo. Tutte queste sequenze possono essere descritte sinteticamente con la notazione:

```
          ┌ Solleva il ricevitore
          │ Componi il numero
n volte  ─┤ (Nessuno risponde)
          └ Deponi il ricevitore
            Solleva il ricevitore
            Componi il numero
            (Qualcuno risponde)
            Conduci la conversazione
```

dove n può essere 0, 1, 2, ...
A queste infinite sequenze va aggiunta la sequenza di lunghezza infinita corrispondente al caso in cui l'interlocutore non risponda mai al telefono. L'algoritmo cioè può evocare un processo ciclico che non avrà mai termine:

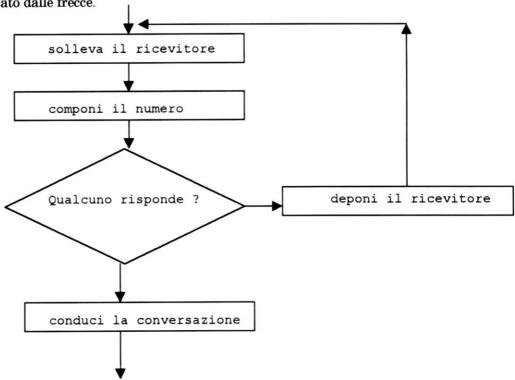
```
infinite { Solleva il ricevitore
  volte  { Componi il numero
         { (Nessuno risponde)
         { Deponi il ricevitore
```

1.3 La specificazione di un algoritmo.

Un algoritmo, dunque, è un testo, cioè un insieme finito di simboli, in grado di descrivere un insieme, in generale infinito, di sequenze di esecuzione o processi sequenziali. Diversi costrutti linguistici possono essere usati per scrivere un algoritmo. Vediamo alcuni metodi possibili nel caso dell'algoritmo per effettuare una telefonata.

Primo metodo.

```
1  Solleva il ricevitore
2  Componi il numero
3  SE qualcuno risponde ALLORA SALTA AL PUNTO 6
4  Deponi il ricevitore
5  TORNA AL PUNTO 1
6  Conduci la conversazione
```

Secondo metodo. Si usa una notazione grafica di uso corrente (flowchart) in cui l'ordine di esecuzione delle operazioni è indicato dalle frecce.

Terzo metodo.

```
Solleva il ricevitore
Componi il numero
SE qualcuno risponde ALLORA Conduci la conversazione
      ALTRIMENTI  Deponi il ricevitore
                  Solleva il ricevitore
                  Componi il numero
                  PROSEGUI IL PROCEDIMENTO A PARTIRE DALLA PAROLA "SE"
```

Terzo metodo (variante).

```
Solleva il ricevitore
Componi il numero
FINO A CHE qualcuno risponde RIPETI LE SEGUENTI OPERAZIONI:
            Deponi il ricevitore
            Solleva il ricevitore
            Componi il numero
POI: Conduci la conversazione
```

Tutti questi algoritmi, se si prescinde dalle "istruzioni di controllo" (ad es., TORNA AL PUNTO 1, PROSEGUI IL PROCEDIMENTO A PARTIRE DALLA PAROLA "SE", ecc.) che servono a guidare l'esecutore durante la scansione del testo dell'algoritmo stesso, descrivono lo stesso insieme di sequenze di esecuzione. La scelta di una notazione o dell'altra non può però essere lasciata al caso. È necessario concordare su un linguaggio adeguato che permetta di rappresentare algoritmi in modo "naturale" rendendo meno difficili a chi si occupa di programmazione le tre operazioni di scrivere, comprendere, comunicare algoritmi.

1.4 Istruzioni e schemi di controllo.

Negli algoritmi finora descritti si possono individuare due classi fondamentali di costrutti linguistici:

- istruzioni o comandi, cioè costrutti come "Deponi il ricevitore" che prescrivono l'esecuzione di determinate operazioni;
- schemi o istruzioni o strutture di controllo, cioè costrutti come "TORNA AL PUNTO 1" che indicano all'esecutore l'ordine in cui tali operazioni devono essere eseguite.

Per evitare ambiguità d'ora in poi le istruzioni verranno sempre rappresentate in caratteri maiuscoli. Sarà inoltre necessario concordare un insieme di regole grammaticali, una sintassi, cioè, che permetta di stabilire con certezza se una data sequenza di caratteri sia da considerarsi o no una istruzione.

1.5 Istruzioni elementari e istruzioni non elementari.

Una istruzione ha lo scopo di essere eseguita. Perché ciò sia possibile, è necessario che l'esecutore dell'algoritmo in cui essa appare ne conosca non solo la sintassi, cioè la sappia identificare, ma conosca anche il significato. Diciamo elementari quelle istruzioni il cui significato possa esser ritenuto noto al loro esecutore, non elementari le altre. Il significato di una istruzione non elementare dovrà essere in qualche modo specificato al suo esecutore in termini di operazioni elementari.
Il fatto che una istruzione sia considerata elementare oppure no dipende dal particolare esecutore a cui essa è rivolta. Ad esempio, l'istruzione:

```
LAVORA UNA MAGLIA BASSA
```

sarà considerata elementare soltanto se l'esecutore a cui essa è diretta abbia esperienza di lavoro all'uncinetto. In caso contrario, il suo significato dovrà essere definito in termini di operazioni più semplici.
Consideriamo come esempio l'istruzione:

```
COMPONI IL NUMERO
```

già utilizzata nell'algoritmo per effettuare una telefonata. Questa potrà essere specificata in termini di istruzioni più semplici, ad esempio così:

```
"COMPONI IL NUMERO"
```

significa:

```
SE LA CHIAMATA È EXTRA-URBANA ALLORA COMPONI IL PREFISSO
POI   COMPONI LA PRIMA CIFRA
      COMPONI LA SECONDA CIFRA
      COMPONI LA TERZA CIFRA
      COMPONI LA QUARTA CIFRA
      COMPONI LA QUINTA CIFRA
      COMPONI LA SESTA CIFRA
```

in cui sottolineati sono i costrutti di controllo.
Ancora si potrà specificare che:

```
"COMPONI IL PREFISSO"
```

significa:

```
COMPONI LA PRIMA CIFRA DEL PREFISSO
COMPONI LA SECONDA CIFRA DEL PREFISSO
```

e così via.

1.6 Variabili ed istruzioni di assegnamento.

Pensiamo di voler specificare un algoritmo per effettuare il prodotto di due numeri interi per addizioni successive. L'algoritmo potrebbe essere costituito semplicemente dal testo seguente:

```
SOMMA IL MOLTIPLICANDO A SE STESSO UN NUMERO DI VOLTE
      UGUALE AL VALORE DEL MOLTIPLICATORE
```

Volendo specificare con maggiore dettaglio le operazioni richieste, si dovrà adottare delle cautele per evitare costrutti ambigui. Si potrà scrivere:

```
SOMMA IL MOLTIPLICANDO A SE STESSO
DECREMENTA DI UNO IL VALORE DEL MOLTIPLICATORE
SOMMA ANCORA IL MOLTIPLICANDO AL VALORE OTTENUTO DALLA PRECEDENTE SOMMA
DECREMENTA DI NUOVO IL VALORE OTTENUTO DALLA PRECEDENTE SOTTRAZIONE
RIPETI IL PROCEDIMENTO
       FINO A CHE PER DECREMENTI SUCCESSIVI NON RAGGIUNGERAI IL VALORE UNO
```

Per evitare ambiguità nel testo si sono usate le espressioni

```
VALORE OTTENUTO DALLA PRECEDENTE SOMMA
```

e

```
VALORE OTTENUTO DALLA PRECEDENTE SOTTRAZIONE
```

per specificare, di volta in volta, il valore considerato.
Più sinteticamente, per distinguere tali valori, si potevano usare dei simboli, o identificatori, che permettessero di volta in volta il riferimento senza ambiguità al valore voluto:

```
CHIAMA INIZIALMENTE M IL VALORE DEL MOLTIPLICANDO ED
                    N IL VALORE DEL MOLTIPLICATORE
RIPETI LE SEGUENTI OPERAZIONI FINO A CHE
                  IL VALORE DI N NON DIVENTI UGUALE A 1
      SOMMA IL VALORE DEL MOLTIPLICANDO AL VALORE DI M
      CHIAMA IL RISULTATO ANCORA M
      SOTTRAI 1 DAL VALORE DI N
      CHIAMA IL RISULTATO ANCORA N
ALLA FINE IL VALORE DI M È IL RISULTATO
```

I simboli M, N dell'esempio si dicono identificatori di variabili. Una variabile può essere immaginata come una scatola dotata di nome, l'identificatore della variabile, e contenente un valore. Il valore di una variabile potrà essere modificato con una istruzione di assegnamento come:

```
ASSEGNA AD N IL VALORE "1"
```

o più sinteticamente:

```
N = 1
```

Tale istruzione inserisce il valore 1 nella scatola di nome N. Il contenuto precedente di N verrà quindi perso.
A destra del simbolo di assegnamento "=" potremo scrivere anche un nome di variabile oppure una espressione complessa:

```
N = M
N = (M + A) * 5 / M
```

Nel secondo caso l'istruzione ha significato soltanto se i valori di M, A ed N sono tali che per essi siano definite le operazioni di somma, moltiplicazione e divisione. Potrebbero essere numeri interi o numeri reali, ad esempio; se una di esse fosse invece una stringa di caratteri, l'istruzione non avrebbe senso.
In generale, sarà conveniente associare ad ogni variabile un tipo ben preciso, cioè specificare la classe di valori che essa è destinata ad assumere.
Se le variabili A, B e C sono di tipo NUMERI INTERI, è corretto l'assegnamento:

```
A = B + C
```

mentre non ha senso l'assegnamento:

```
A = PRIMA_LETTERA(B)
```

dove con PRIMA_LETTERA(X) si definisce l'operazione di estrazione del primo carattere dalla stringa x.
Se A è di tipo NUMERI INTERI e B di tipo NUMERI REALI, sarà in generale considerato scorretto l'assegnamento

```
A = B
```

1.7 Strutture di dati.

Spesso si ha la necessità di dover individuare non un singolo valore, ma un'intera classe di valori. Nel linguaggio naturale si ricorre in tale caso a nomi collettivi e a particolari costrutti che specificano una "legge di selezione" del singolo elemento della classe, qualora ad esso ci si debba riferire.
Una tecnica analoga può essere convenientemente usata anche nel linguaggio di descrizione degli algoritmi. Si useranno in questo caso nomi di "variabili collettive" che assumono non più un singolo valore ma una n-pla di valori. Tutte le volte che si usa una variabile collettiva si devono definire rigorosamente quali operazioni si hanno a disposizione per selezionare, tra gli n valori associati alla variabile, un valore ben preciso.
Ad esempio, sia data la variabile collettiva MAZZO_DI_CARTE, il cui valore sarà costituito dall'insieme dei 52 valori che rappresentano le 52 carte del mazzo. Potremmo supporre di avere a disposizione una operazione che selezioni la i-esima carta del mazzo, supposto ordinato, per ogni i fra 1 e 50:

```
I_ESIMA(MAZZO_DI_CARTE)
```

oppure, supporre di avere a disposizione una operazione che permetta di selezionare la prima carta del mazzo:

```
TESTA(MAZZO_DI_CARTE)
```

o ancora, una operazione che permette di estrarre una carta qualsiasi dal mazzo:

```
ESTRAI(MAZZO_DI_CARTE)
```

In generale, si parlerà di strutture di dati ogniqualvolta si abbia a disposizione un insieme di valori descritti da un nome collettivo, e un insieme di funzioni di accesso o operazioni di selezione, che ci permettano di individuare un ben preciso elemento dell'insieme di valori.

1.7.1 Array.

La struttura array può pensarsi come un insieme ordinato di variabili, ciascuna delle quali è individuabile con un indice, cioè con un valore intero che specifica la sua posizione nell'insieme.
Con la scrittura

```
MAZZO_DI_CARTE(3)
```

indicheremo l'elemento dell'array MAZZO_DI_CARTE con indice 3, cioè la terza carta del mazzo.
Dato un array NUMERI costituito da 100 componenti intere, con

```
NUMERI(5) = 10
```

si intenderà:

```
SELEZIONA LA QUINTA VARIABILE DI NUMERI
ASSEGNA AD ESSA IL VALORE 10
```

Si può pensare anche ad array con più di un indice. L'uso di 2 indici permette ad esempio facilmente di rappresentare tabelle: il primo individua la riga in cui si trova l'elemento da reperire, il secondo ne individua la colonna.
Array ad un indice sono detti anche vettori. L'uso di array permette spesso una chiara e compatta descrizione di algoritmi in cui vengono ripetute operazioni analoghe.
Si consideri l'algoritmo per formare un numero telefonico:

```
FORMA LA PRIMA CIFRA
FORMA LA SECONDA CIFRA
FORMA LA TERZA CIFRA
FORMA LA QUARTA CIFRA
FORMA LA QUINTA CIFRA
FORMA LA SESTA CIFRA
```

Esso può essere riscritto inserendo le cifre del numero telefonico in un vettore CIFRA:

```
FACENDO VARIARE I DA 1 A 6 RIPETI
      FORMA CIFRA(I)
```

Accanto alla maggior compattezza di scrittura si possono così ottenere anche altri vantaggi. Trasformando l'algoritmo così:

```
ASSEGNA 6 ALLA VARIABILE K
FACENDO VARIARE I DA 1 A K RIPETI
      FORMA CIFRA(I)
```

si ha un programma più generale che permette, cambiando l'istruzione di assegnamento, di formare numeri telefonici composti da differenti numeri di cifre.

1.7.2 Coda.

Anche la struttura coda descrive un insieme ordinato di variabili, le funzioni d'accesso del quale permettono esclusivamente di estrarre il primo elemento nella coda o di aggiungere uno nuovo dopo l'ultimo. Una coda è quindi una struttura che viene "consumata" a partire sempre dal primo elemento, o "generata" sempre dopo l'ultimo.

1.7.3 File.

La struttura file descrive un insieme organizzato come una coda, ma che è possibile esclusivamente "consumare" oppure esclusivamente "generare". Si parlerà nel primo caso di file di ingresso, in quanto le variabili verranno reperite una per una durante l'esecuzione dell'algoritmo per essere elaborate, mentre si parlerà nel secondo caso di file d'uscita, in quanto le successive generazioni per accodamento produrranno un file come risultato dell'elaborazione.
Un mazzo di carte da distribuire tra quattro giocatori può essere considerato un esempio di file d'ingresso, operato mediante l'algoritmo:

```
PRELEVA LA PRIMA CARTA DISPONIBILE DEL MAZZO
CONSEGNALA AL PRIMO GIOCATORE
PRELEVA LA PRIMA CARTA DISPONIBILE DEL MAZZO
CONSEGNALA AL SECONDO GIOCATORE
PRELEVA LA PRIMA CARTA DISPONIBILE DEL MAZZO
CONSEGNALA AL TERZO GIOCATORE
PRELEVA LA PRIMA CARTA DISPONIBILE DEL MAZZO
CONSEGNALA AL QUARTO GIOCATORE
RIPETI LE PRECEDENTI OPERAZIONI FINO ALLA FINE DEL MAZZO
```

1.7.4 Stack.

La struttura stack descrive un insieme di variabili su cui è possibile operare esclusivamente mediante le seguenti tre funzioni d'accesso:
- PUSH che aggiunge un elemento allo stack;
- POP che toglie allo stack l'elemento che è stato aggiunto per ultimo;
- TOP che rende disponibile il valore dell'elemento che è stato aggiunto per ultimo.

1.8 Costrutti di controllo.

In generale, si diranno costrutti di controllo quei costrutti che specificano in quale ordine le istruzioni devono essere eseguite.
I costrutti di controllo utilizzabili per la specificazione di un algoritmo in italiano possono essere di natura assai diversa:
- costrutti di sequenziazione, come: PER PRIMA COSA, DOPO DI CIÒ, POI, ecc.
- costrutti condizionali o di selezione, con i quali la esecuzione di una certa operazione viene subordinata al verificarsi o meno di una specificata condizione. Ad esempio, costrutti come SE...ALLORA o forme analoghe come NEL CASO CHE, ALLORA, ALTRIMENTI, ecc.
- costrutti iterativi, che prescrivono di ripetere un certo numero di volte la esecuzione di un gruppo di operazioni. Ad esempio, costrutti come E SI RIPETE IL PROCEDIMENTO. QUESTO AVVIENE TANTE VOLTE FINCHÉ oppure RIPETERE DA ° PER 5 VOLTE o ancora IL PROCEDIMENTO CONTINUA FINO A CHE.

I costrutti di sequenziazione, selezione e iterazione possono essere considerati i tre fondamentali "modelli di pensiero" mediante i quali noi siamo in grado di descrivere qualunque successione di eventi, cioè qualunque processo sequenziale.

2. Le strutture di controllo.

2.1 Sequenza.

Il costrutto di controllo più semplice è quello che specifica che due o più operazioni (elementari o no) devono essere eseguite una dopo l'altra.
La notazione convenzionale di tale costrutto è:

```
segmento_1
segmento_2
……………..
segmento_n
```

oppure:

```
BEGIN
      segmento_1
      segmento_2
      ……………..
      segmento_n
END
```

Con il termine "segmento" si indica un generico costrutto che prescrive la esecuzione di una o più operazioni elementari o no.

2.2 Selezione binaria.

La struttura di selezione più semplice è quella fra due sole alternative, del tipo "se… allora… altrimenti…", che rappresenteremo con il costrutto:

```
IF condizione      THEN segmento_1
                   ELSE segmento_2
ENDIF
```

Il suo significato è il seguente: viene dapprima valutata la condizione specificata, la quale può essere o vera o falsa. Nel primo caso, verrà eseguito il segmento_1; nel secondo il segmento_2. In entrambi i casi, dopo l'esecuzione del segmento selezionato, il controllo verrà trasferito alla istruzione fisicamente successiva alla clausola di chiusura ENDIF.
Ad esempio, la scrittura:

```
IF IL TELEFONO È LIBERO THEN CONDUCI LA CONVERSAZIONE
                        ELSE DEPONI IL RICEVITORE
ENDIF
```

evoca le seguenti sequenze di esecuzione:

```
IL TELEFONO È LIBERO (vero)
CONDUCI LA CONVERSAZIONE

IL TELEFONO È LIBERO (falso)
DEPONI IL RICEVITORE
```

La clausola:

```
ELSE segmento_2
```

è opzionale. Se essa non viene specificata, e se la condizione risulta falsa, il controllo viene trasferito senz'altro alla istruzione successiva a ENDIF.

2.3 Selezione n-aria.

La struttura "se... allora... altrimenti..." può essere generalizzata al caso in cui le alternative siano più di due.
In tal caso si usa la seguente notazione:

```
IF condizione_1         THEN segmento_1
ELSEIF condizione_2     THEN segmento_2
ELSEIF condizione_3     THEN segmento_3
.........
ELSEIF condizione_(n-1) THEN segmento_(n-1)
                        ELSE segmento_n
ENDIF
```

dove n è maggiore di 2.
Tale scrittura ha il seguente significato. Si valuta dapprima la sola condizione_1: se essa risulta vera, viene eseguito il segmento_1, quindi il controllo viene trasferito alla istruzione che segue ENDIF.
Se, viceversa, la condizione_1 risulta falsa, si valuta la (sola) condizione_2, e così via.
Se tutte le n-1 condizioni risultano false, si esegue il segmento_n (clausola ELSE).
Osserviamo che non si è imposto che le condizioni specificate siano mutuamente esclusive: ciò non crea ambiguità, poiché le condizioni vengono valutate una alla volta nell'ordine specificato.
Anche in questo caso si conviene che la clausola:

```
ELSE segmento_n
```

possa essere tralasciata. In questo caso se tutte le condizioni risultano false, il controllo viene trasferito senz'altro alla istruzione successiva a ENDIF.
L'uso di questo costrutto è illustrato dai seguenti esempi autoesplicativi.
Esempio 1.

```
IF          COLORE == ROSSO   THEN PREZZO = 100
ELSEIF      COLORE == VERDE   THEN PREZZO = 150
ELSEIF      COLORE == GIALLO  THEN PREZZO = 200
ELSEIF      COLORE == BLU     THEN PREZZO = 250
                              ELSE PREZZO = 300
ENDIF
```

Esempio 2.

```
ESAMINA I DATI ANAGRAFICI DELLA PROSSIMA PERSONA
IF CITTÀ DI ORIGINE == MILANO
            THEN INCREMENTA CONTATORE MILANESI
ELSEIF REGIONE DI ORIGINE == LOMBARDIA
            THEN INCREMENTA CONTATORE LOMBARDI
ELSEIF PAESE DI ORIGINE == ITALIA
            THEN INCREMENTA CONTATORE ITALIANI
            ELSE INCREMENTA CONTATORE PERSONE
ENDIF
```

2.4 Selezione per casi.

Un'altra struttura di selezione assai frequente è quella per casi, che può essere espressa in italiano con locuzioni del tipo: "si consideri x; possono darsi i seguenti casi: se x è a, allora..., se x è b, allora...".

Si userà la seguente notazione:

```
TEST espressione
      CASE lista_1 di casi
            segmento_1
      CASE lista_2 di casi
            segmento_2
      ……………..
      OTHERWISE
            segmento_n
ENDTEST
```

dove le liste di casi sono liste di espressioni (separate da virgole) dello stesso tipo della espressione sotto TEST (intere se essa ha valore intero, alfanumeriche se essa ha valore alfanumerico, ecc.).
Il significato della scrittura è il seguente. Si valuta dapprima la espressione indicata alla clausola TEST, quindi si valutano tutte le espressioni della lista_1 di casi. Se il valore di una delle espressioni in lista è uguale al valore della espressione sotto TEST, viene eseguito il segmento_1, quindi il controllo viene trasferito alla prima istruzione successiva alla clausola ENDTEST, e la esecuzione della struttura è terminata. Altrimenti, si valutano tutte le espressioni della lista_2 di casi, e così via. Se tutti i confronti falliscono, viene eseguito il segmento_n (clausola OTHERWISE), dopo di che l'esecuzione della struttura termina.
Anche qui, si conviene che quando la clausola:

```
OTHERWISE segmento_n
```

Non sia specificata, ogni caso non previsto faccia trasferire senz'altro il controllo all'istruzione che segue ENDTEST.
Esempio 1: la già vista selezione per colori.

```
TEST COLORE
      CASE ROSSO
            PREZZO = 100
      CASE VERDE
            PREZZO = 150
      CASE GIALLO
            PREZZO = 200
      CASE BLU
            PREZZO =250
      OTHERWISE
            PREZZO = 300
ENDTEST
```

Esempio 2.

```
TEST REGIONE_DI _PROVENIENZA
      CASE LIGURIA, PIEMONTE, LOMBARDIA, VENETO
            STAMPA "NORD"
      CASE UMBRIA, LAZIO, ABRUZZI
            STAMPA "CENTRO"
      CASE CALABRIA, SICILIA, BASILICATA
            STAMPA "SUD"
      OTHERWISE
            STAMPA "ALTRE REGIONI"
ENDTEST
```

Le liste di casi, in accordo alle convenzioni poste, possono essere liste di variabili. In questo caso, alle variabili che compaiono nelle liste di casi potrebbero essere assegnati, prima dell'ingresso nella struttura di selezione, valori differenti di volta in volta.

Si ammettono anche espressioni complesse, ad esempio:

```
TEST PARTE_INTERA(IMPORTO/1000000)
      CASE 0
            STAMPA "IMPORTO TRA ZERO E UN MILIONE"
      CASE 1, 2, 3
            STAMPA "IMPORTO TRA UNO E QUATTRO MILIONI"
      CASE 4, 5, 6, 7, 8, 9
            STAMPA "IMPORTO TRA QUATTRO E DIECI MILIONI"
      OTHERWISE
            STAMPA "IMPORTO SUPERIORE A DIECI MILIONI"
ENDTEST
```

Nell'esempio la clausola TEST richiede la valutazione di una espressione aritmetica intera.
L'esempio suggerisce una generalizzazione della struttura per casi che permetta la selezione per intervalli numerici. Pertanto, in una lista di casi potranno comparire, oltre a costanti, espressioni del tipo:
(a:b) che significa: "l'intervallo compreso fra a e b inclusi"
:b) che significa: "l'intervallo fra -∞ e b incluso"
(a: che significa: "l'intervallo fra a incluso e + ∞"
a:b che significa: "l'intervallo compreso fra a e b esclusi"
:b che significa: "l'intervallo fra -∞ e b escluso"
a: che significa: "l'intervallo fra a escluso e +∞"
in cui a e b sono espressioni aritmetiche qualsiasi (purché dello stesso tipo della espressione sotto TEST).

2.5 Iterazioni enumerative.

Le iterazioni di tipo più semplice sono quelle in cui si richiede di iterare un dato segmento un numero prefissato di volte. Considereremo due costrutti di questo tipo: la iterazione enumerativa semplice, e la iterazione enumerativa con indice.
Per la prima useremo la seguente notazione:

```
DO n TIMES
      segmento
REPEAT
```

In cui n è una variabile a valori interi, il cui valore non viene mai modificato durante l'esecuzione del segmento.
La struttura specifica che il segmento deve essere eseguito n volte.
Ad esempio, l'algoritmo:

```
PRODOTTO = 0
DO M TIMES
      PRODOTTO = PRODOTTO + N
REPEAT
```

esegue il prodotto di M per N, e lo assegna alla variabile PRODOTTO, per addizioni successive.
Se n è nullo o negativo, il segmento non viene eseguito e quindi l'esecuzione della struttura non ha alcun effetto (il controllo passa all'istruzione successiva).

Le iterazioni enumerative con indice, invece, corrispondono a costrutti del tipo "facendo variare l'indice i a partire da... fino ad arrivare a... esegui...", disponibili nella maggior parte dei linguaggi di programmazione più in uso. In esse, un contatore (detto l'indice o la variabile di controllo della iterazione), viene aggiornato ad ogni ciclo. L'iterazione ha termine quando l'indice raggiunge un valore estremo prefissato.
Si userà la seguente notazione:

```
DO VARYING indice FROM inizio TO fine STEP passo
      segmento
REPEAT
```

Dove l'indice è una variabile a valori interi con segno, e dove inizio, fine e passo sono costanti intere (con segno) oppure variabili a valori interi (con segno). S'impone inoltre che il segmento non contenga istruzioni che modifichino il valore dell'indice, o dell'inizio, o della fine, o del passo.
Il significato nel caso di passo positivo è il seguente: si assegna all'indice, come primo valore, il valore di inizio; quindi si confronta il valore dell'indice con il valore di fine. Se il valore dell'indice è superiore al valore di fine, il controllo passa alla istruzione immediatamente successiva alla clausola REPEAT, e l'esecuzione della struttura è terminata. Altrimenti, viene eseguito il segmento, quindi il valore dell'indice viene incrementato del valore di passo. A questo punto, si effettua di nuovo il confronto con il valore di fine, e così via.

Il significato nel caso di passo negativo è il seguente: la esecuzione procede come nel caso precedente, con la sola differenza che il controllo abbandona la struttura non appena il valore dell'indice diventa inferiore al valore di fine. In questo caso, cioè, l'indice viene fatto scorrere "a ritroso".

Evidentemente non viene analizzato il caso di passo = 0, che non deve essere permesso. Infatti, in tale situazione, l'iterazione non sarebbe mai in grado di raggiungere il valore finale a partire dal valore iniziale, e l'esecuzione continuerebbe all'infinito.
La clausola:

```
STEP passo
```

Può essere tralasciata, nel qual caso si assume che il passo della iterazione sia uguale a +1.
Esempio. La scrittura:

```
DO VARYING I FROM 1 TO NUMERO_DELLE_PERSONE
      ELABORA DATI ANAGRAFICI DELLA I-ESIMA PERSONA
REPEAT
```

evoca la seguente sequenza di computazione:

```
                   I = 1
NUMERO_DELLE_    { I > NUMERO_DELLE_PERSONE (falso)
PERSONE volte    { ELABORA DATI ANAGRAFICI DELLA I-ESIMA PERSONA
                 { I = I + 1
                   I > NUMERO_DELLE_PERSONE (vero)
```

All'uscita dalla struttura il valore di I è NUMERO_DELLE_PERSONE + 1.

2.6 Iterazioni non enumerative.

Le iterazioni non enumerative sono espresse da frasi italiane del tipo "ripeti... fintantoché...", "il procedimento continua finché non..." e simili, cioè quelle in cui il numero di cicli non è in generale noto al momento dell'inizio dell'iterazione, ma dipende dal verificarsi o meno di una specificata condizione. Useremo quattro varianti di questa struttura.

```
DO UNTIL (condizione)
      segmento
REPEAT

DO WHILE (condizione)
      segmento
REPEAT
```

Le due strutture hanno il seguente significato: viene dapprima valutata la condizione. Se questa è vera (rispettivamente, falsa), l'esecuzione della struttura termina, e il controllo viene trasferito all'istruzione fisicamente successiva alla clausola REPEAT. Altrimenti, viene eseguito il segmento e, quindi, viene di nuovo valutata la condizione, e così via.

```
DO
      segmento
REPEAT UNTIL (condizione)

DO
      segmento
REPEAT WHILE (condizione)
```

Il significato di queste due strutture è analogo alle precedenti, con la differenza che il segmento viene sempre eseguito almeno una volta. Più precisamente: viene eseguito il segmento, quindi si valuta la condizione. Se questa è vera (rispettivamente, falsa) l'esecuzione della struttura ha termine. Altrimenti, il segmento viene eseguito di nuovo, e così via.
Esempio.

```
LEGGI UNA SCHEDA
DO UNTIL (SCHEDA LETTA CONTIENE "FINE")
      ELABORA LA SCHEDA
      LEGGI UNA NUOVA SCHEDA
REPEAT
```

L'algoritmo legge ed elabora n schede ($n \geq 1$), di cui l'ultima contiene il messaggio "FINE".
Nel caso in cui fossimo certi della esistenza di almeno una scheda diversa da "FINE", potremo invece scrivere:

```
LEGGI UNA SCHEDA
DO
      ELABORA LA SCHEDA
      LEGGI UNA NUOVA SCHEDA
REPEAT UNTIL (SCHEDA LETTA CONTIENE "FINE")
```

oppure anche:

```
LEGGI UNA SCHEDA
DO
      ELABORA LA SCHEDA
      LEGGI UNA NUOVA SCHEDA
REPEAT WHILE (NON SCHEDA LETTA CONTIENE "FINE")
```

Osserviamo che, nelle quattro strutture indicate, il segmento da iterare deve essere tale da poter modificare il valore di verità della condizione. Se così non fosse, infatti, una volta iniziata l'iterazione, questa non avrebbe mai fine. Una condizione necessaria (ma non sufficiente) è che il segmento da iterare modifichi il valore di almeno una delle variabili su cui la condizione è espressa.

2.7 Iterazioni a più uscite.

Le strutture iterative finora introdotte non permettono di trattare agevolmente alcune situazioni assai frequenti nella programmazione. Una situazione di tal genere si ha ad esempio nel caso di algoritmi di ricerca tabellare, in cui un elemento (ELEMENTO_CERCATO) viene confrontato via via con tutti gli elementi presenti in una tabella, fino a quando o l'elemento venga trovato o vengano esauriti senza successo tutti gli elementi della tabella.
Se non si richiede di distinguere i due casi (trovato e non trovato), l'algoritmo può essere espresso semplicemente nella forma:

```
AZZERA I
DO
      INCREMENTA I
REPEAT UNTIL (I > NUMERO_ELEMENTI_TABELLA OPPURE
            ELEMENTO_CERCATO == I_ESIMO_ELEMENTO_TABELLA)
```

ma se i due casi devono essere distinti, è necessario duplicare un test, aggiungendo, all'uscita della struttura, una selezione binaria, ad esempio:

```
......
IF I > NUMERO_ELEMENTI_TABELLA      THEN STAMPA "NON TROVATO"
                                    ELSE STAMPA "TROVATO"
```

Per esprimere un tale tipo di situazioni si userà la seguente notazione:

```
LOOP
      segmento_1
      EXIT IF (condizione_1) AND DO segmento_2
      segmento_3
      EXIT IF (condizione_2) AND DO segmento_4
      ...........
      segmento_(2n-1)
      EXIT IF (condizione_n) AND DO segmento_2n
      segmento_(2n+1)
ENDLOOP
```

In cui tutte le clausole

```
AND DO segmento_i
```

sono opzionali e in cui almeno una clausola EXIT IF deve essere presente.
Il significato della scrittura è il seguente. Viene eseguito il segmento_1, quindi viene valutata la condizione_1. Se questa risulta vera, allora viene eseguito il segmento_2, quindi l'esecuzione della struttura ha termine (cioè il controllo viene trasferito alla istruzione successiva alla clausola ENDLOOP). Se la condizione_1 è falsa, si prosegue in modo analogo con il segmento_3. Dopo la esecuzione del segmento_(2n+1), il procedimento viene ripreso da capo.

Esempio 1. Elaborazione di un pacco di schede (riscrittura dell'algoritmo già esaminato in precedenza).

```
LOOP
      LEGGI UNA SCHEDA
      EXIT IF (SCHEDA LETTA CONTIENE "FINE")
      ELABORA LA SCHEDA
ENDLLOP
```

Esempio 2. Ricerca tabellare.

```
AZZERA I
LOOP
      INCREMENTA I
      EXIT IF I > NUMERO_ELEMENTI_TABELLA AND DO STAMPA "NON TROVATO"
      EXIT IF ELEMENTO_CERCATO == I_ESIMO_ELEMENTO DELLA_TABELLA
               AND DO STAMPA "TROVATO"
ENDLOOP
```

2.8 Nidificazione di strutture

Le strutture di controllo possono essere composte le une con le altre secondo una modalità che chiameremo di innestamento (nesting) o nidificazione.
Ad esempio, nella struttura:

```
IF (condizione_1) THEN segmento_1 ENDIF
```

il segmento_1 potrebbe essere semplicemente una istruzione elementare, oppure a sua volta una struttura, ad esempio una sequenza di due segmenti:

```
IF (condizione_1) THEN  BEGIN
                              segmento_1.1
                              segmento_1.2
                        END
ENDIF
```

i quali potrebbero a loro volta essere costituiti, rispettivamente, da una istruzione elementare e da una iterazione:

```
IF (condizione_1) THEN  BEGIN
                              istruzione_1
                              DO VARYING I FROM 1 TO 100
                                    segmento_1.2.1
                              REPEAT
                        END
ENDIF
```

e così via.
Un programma viene così ad essere organizzato in modo gerarchico: esso sarà costituito da una struttura di controllo di livello più alto, contenente una o più strutture di controllo in essa nidificate, e così via fino a raggiungere il livello delle istruzioni elementari. Si noti che la composizione di strutture per innestamento è possibile per il fatto che esse possiedono tutte un unico "punto di ingresso" e un unico "punto di uscita".

3. Schemi di controllo guidati da eventi.

Considereremo due costrutti linguistici, uno di tipo selettivo e uno di tipo iterativo, non ancora disponibili nei linguaggi di programmazione più in uso, che sono, almeno da un punto di vista concettuale, assai differenti dai costrutti precedenti. Mentre quelli erano di tipo strettamente imperativo ("fai questo nel caso che..."), questi comprendono anche aspetti di tipo dichiarativo ("se si verifica questo, allora vuol dire che..."). In altre parole, in essi ci si preoccupa, in primo luogo, di individuare una certa situazione (l'occorrenza di un certo evento); soltanto in seguito verrà specificato che cosa fare a causa di quella situazione.

3.1 Selezione guidata da eventi.

Per comprendere meglio il senso di questo schema si esamini il seguente esempio informale in italiano di un algoritmo per effettuare una telefonata:

```
      ┌ SOLLEVA IL RICEVITORE
      │ COMPONI IL NUMERO
      │ ASCOLTA IL SEGNALE
(a)  <  SE SENTI UN SUONO PROLUNGATO RIPETUTO ALLORA ÈLIBERO
      │ SE SENTI UN SUONO BREVE RIPETUTO ALLORA È OCCUPATO
      └ SE NON SENTI ALCUN SUONO ALLORA C'È QUALCHE GUASTO
```

```
        QUANDO È LIBERO
              ATTENDI LA RISPOSTA
              CONDUCI LA CONVERSAZIONE
              DEPONI IL RICEVITORE
(b)     QUANDO È OCCUPATO
              DEPONI IL RICEVITORE
        QUANDO C'È QUALCHE GUASTO
              DEPONI IL RICEVITORE
              CHIAMA IL TECNICO
```

Nell'esempio si possono individuare due parti ben distinte, messe in evidenza dalle graffe. La prima parte (a) è costituita da un singolo segmento, che chiameremo segmento di selezione. Questo contiene alcune frasi che segnalano la occorrenza di "eventi": È LIBERO, È OCCUPATO, C'È QUALCHE GUASTO.

La seconda parte (b) è costituita da più segmenti, uno per ogni diverso nome di evento presente nella prima parte.

Il meccanismo di selezione è il seguente: viene eseguita la prima parte, fino a quando non venga incontrata la segnalazione di un evento. Allora l'esecuzione della prima parte termina e il segmento corrispondente all'evento segnalato viene eseguito.

Ad esempio, all'occorrenza dell'evento OCCUPATO, verrà selezionato il segmento associato alla clausola QUANDO È OCCUPATO: cioè la singola istruzione DEPONI IL RICEVITORE. Quindi l'esecuzione dell'algoritmo termina.

Per rappresentare questa struttura useremo la seguente notazione:

```
EXECUTE UNLESS (nome_evento_1, …, nome_evento_n)
            segmento di selezione
      WHEN nome_evento_1
      segmento_1
      WHEN nome_evento_2
      segmento_2
      ..............
      WHEN nome_evento_n
      segmento_n
END EXECUTE
```

in cui il verbo EXECUTE individua il punto di ingresso nella struttura, e la clausola END EXECUTE ne individua il punto di uscita.

La clausola:

```
UNLESS (nome_evento_1, …, nome_evento_n)
```

lista i nomi di tutti gli eventi che possono essere segnalati nel segmento di selezione, facilitandone quindi l'identificazione.

La segnalazione di un evento verrà effettuata, nel segmento di selezione, con la frase:

```
EVENT nome_evento
```

oppure, se la segnalazione di un evento è subordinata al verificarsi di una certa condizione, con la frase:

```
EVENT nome_evento IF condizione
```

Il significato della struttura più su esposta è il seguente. Il controllo viene inizialmente trasferito alla prima istruzione del segmento di selezione, che può contenere istruzioni di tipo qualsiasi. L'esecuzione di una frase EVENT nome_evento_i o di una frase EVENT nome_evento_i IF condizione, in cui la condizione risulti vera, fa sì che il controllo venga trasferito al segmento_i che segue la clausola WHEN nome_eventi_i. Alla fine della esecuzione di tale segmento, il controllo viene trasferito alla prima istruzione dopo END EXECUTE.

Se l'esecuzione del segmento di selezione termina senza che nessun evento venga segnalato, il controllo passa senz'altro alla prima istruzione successiva a END EXECUTE.

Un esempio è il seguente algoritmo informale per cambiare un assegno:

```
EXECUTE UNLESS (ASSEGNO_VALIDO, QUALCHE_DATO_MANCANTE,
                  IMPORTO_SCOPERTO, FIRMA_SCONOSCIUTA)
      EVENT QUALCHE_DATO_MANCANTE
            IF FIRMA O IMPORTO O BENEFICIARIO O DATA MANCANTI
      CONFRONTA FIRMA ASSEGNO CON FIRMA DEPOSITATA
      EVENT FIRMA_SCONOSCIUTA IF FIRMA NON CORRISPONDENTE
      CALCOLA SALDO CONTO CORRENTE
      EVENT IMPORTO_SCOPERTO IF IMPORTO ASSEGNO > SALDO
      EVENT ASSEGNO_VALIDO
      WHEN ASSEGNO_VALIDO
            CAMBIA ASSEGNO
      WHEN QUALCHE_DATO_MANCANTE
            RESTITUISCI ASSEGNO
      WHEN IMPORTO_SCOPERTO
            REGISTRA ESTREMI ASSEGNO
            RESTITUISCI ASSEGNO
      WHEN FIRMA_SCONOSCIUTA
      INTERPELLA DIRETTORE
END EXECUTE
```

3.2 Iterazione interrotta da eventi.

L'idea di introdurre il concetto di evento per collegare il codice che rileva l'occorrenza di una determinata condizione al codice adibito al suo trattamento, può essere applicata per generalizzare la struttura iterativa a più uscite introdotta precedentemente. Ciò si può fare in sostanza sostituendo le varie clausole del tipo:

```
EXIT IF (condizione) AND DO segmento
```

con delle segnalazioni di eventi, e "raccogliendo" il codice dei vari segmenti che compaiono nelle sottoclausole AND DO... nella porzione finale del costrutto. Useremo pertanto la seguente notazione:

```
LOOP UNLESS (nome_evento_1, …, nome_evento_n)
            segmento da iterare
      WHEN nome_evento_1
            segmento_1
      WHEN nome_evento_2
            segmento_2
      …………….
      WHEN nome_evento_n
            segmento_n
END LOOP
```

in cui il segmento da iterare conterrà, come nella selezione guidata da eventi, n clausole del tipo:

```
EVENT nome_evento
```

oppure:

```
EVENT nome_evento IF condizione
```

Il significato della scrittura introdotta è del tutto analogo a quello di una struttura EXECUTE in cui il segmento di selezione sia così definito:

```
DO UNTIL (FALSO)
      segmento da iterare
REPEAT
```

cioè in cui il segmento di selezione sia costituito da un ciclo infinito (FALSO è la condizione sempre falsa), che può essere interrotto solo dalla segnalazione di un evento.

4. Bibliografia.

G.A. Lanzarone, M. Maiocchi, R. Polillo: Introduzione alla programmazione strutturata. - Franco Angeli Editore, Milano 1979.